

TO WHOM IT MAY CONCERN, THE FOLLOWING IS  
A SPECIFICATION OF THE AFORESAID INVENTION

**MONIKER METHOD, APPARATUS, SYSTEM**  
**AND ARTICLE OF MANUFACTURE**

**BACKGROUND OF THE INVENTION**

**Field of the Invention.**

The present invention relates to a method, system, apparatus and an article of manufacture for accessing objects or more particularly, to a moniker for locating, activating and (or) initializing specific instances of program objects particularly in a large computer system environment such as a network and (or) particularly in an environment where the object is not registered, i.e., does not have a persistent state.

**Related Information.**

A *moniker* is a "smart" name which contains information that is used to link to instances of other objects. Monikers are **Object Oriented Programming** (OOP) programs and a better appreciation of them will be realized from the following general discussion

Object-Oriented Programming (OOP) is based on the ideology of using independent modular programs, called **objects**, as the building blocks for programming applications. The programming applications themselves may be considered composites of these objects. OOP prefers to think of a programming application as a **client** that invokes a particular object program, known as a **class**, as an **instance** of that object. The modularity of programming objects allows them to be **instantiated**, i.e., invoked, a plurality of times to create several run-time versions, i.e., objects, of the same object program. And therein lies the power of OOP – pieces of software (i.e., the object

programs) are used, reused and interchanged between programming applications thereby avoiding redundancy, maintaining efficiency and freeing programmers to focus their attentions on the **kernel**, or core, of the programming application.

5        **Windows NT™** (Windows New Technology) is a common operating system (O/S) from Microsoft™ that supports the OOP methodology. To support this methodology, Windows NT™ must provide a platform for executing OOP applications and accessing objects. In order to interface OOP application programs to the Windows NT™ operating platform, the O/S provides a set of functions, also referred to as  
10        methods, called the **Application Programming Interface (API)**. The API is a language message format used by an application program to communicate with the Windows NT™ operating system (or other system program such as a database management system). APIs are implemented by writing function calls in the program that provide the linkage to a specific subroutine for execution.

15        **Component Object Model (COM)** is the component software architecture that the Windows NT™ operating system employs to access objects. Accessing objects in a common way on a system is paramount when one considers that objects, according to OOP methodology, are independent from the client. Thus, objects may be located  
20        anywhere on the system including program applications or persistent memory. To effect a common accessing scheme, COM defines a set structure for building program routines (objects) that can be called up and executed in the Windows™ environment. COM itself is written in an OOP language and objects therein are known as COM objects.

25        Programmers have evolved COM into a compound document technology known as **Object Linking and Embedding (OLE)** to handle the complex task accessing or

embedding objects in applications or documents, called the container application. OLE, for example, allows an object such as a spreadsheet or video clip to be embedded into a document, called the container application. When the object is double clicked, the application that created it, called the **server** application, is launched in order to edit it.

5 An object can be linked instead of embedded -in which case the container application does not physically hold the object, but provides a pointer to it. If a change is made to a linked object, all the documents that contain that same link are automatically updated the next time the user opens them. An application can be both client and server.

10 The present invention relates to accessing objects within the COM environment.

In order to instantiate a COM class, the client must first grab hold of a pointer to a COM-compliant interface. The set of operations carried out to somehow retrieve a valid interface pointer to a live object is called **binding**. In the case where the client needs only a general instance of the class, this interface pointer is obtained by creating a new instance of the co-class through the well known **CoCreateInstance()** API provided by OLE, as demonstrated in the following code snippet:

```

Interface* NewObject(clsid)
{
20  HRESULT hr;
    Interface* pInterface = NULL;
    // Error handling omitted
    hr = CoCreateInstance(clsid, NULL, CSLCTX_ALL, IID_IInterface,
        (void**)&pInterface);
25  if (SUCCEEDED(hr))
        return pInterface;
    return NULL;

```

The function CoCreateInstance creates an instance of the identified class and receives a pointer to the IDispatch interface of that instance. The IDispatch interface allows for the invocation of methods that are bound to at run time.

5 The foregoing interface works well for instantiating objects. However, when the client needs to access an already – existing object (a specific instance), the CoCreateInstance function is not applicable. For example, in the case where an Excel™ application (client) links to a pre-stored Excel™ data file (specific instance), it will not do to instantiate a new object. Instantiating a new object creates a new version of that object – it does not access the specific instance.

Moreover, it is undesirable to employ the CoCreateInstance function because it requires the server to be actively involved in instantiating the relevant object. Recalling that the tenet of Object Oriented Programming is to establish independency between programs, it is unsatisfactory that the server be burdened with calling and passing parameters to the instantiation routine.

COM provides special programming objects called monikers that allow clients, such as an Excel™ application, to link specific instances of an object. In simplistic terms, a moniker is a smart name which stores that information which allows the client to locate and invoke the instance. In our Excel™ example, a call to the moniker that points to the Excel™ data file will automatically produce a pointer to the parent Excel™ application. In this case, the operating system will automatically launch the Excel™ application and link it to the Excel™ data file. The Excel™ application need not be concerned about the details of linking to the object.

Monikers while useful, are limited because they operate within the COM environment. In particular, COM monikers call only those objects that are registered in

the operating system. This is a problem because the objects themselves are responsible for registration. In the case where, for example, a peripheral element such as a PLC (Programmable Logic Controller) is connected to a personal computer (PC), the PLC has no conventional way in which to register itself with the operating system.

5 In that case, the standard monikers are unable to provide a pointer to the clients of the operating system for linking to the PLC. This is particularly true for a PLC that is remotely connected.

Another problem is that monikers automatically instantiate the object whether or not the object is already running. In the PLC <sup>TM</sup> example, the moniker would cause the PLC to be automatically instantiated. This is very dangerous because it may cause an otherwise dormant PLC to come "alive" and activate machinery connected thereto. This could have disastrous results in a manufacturing environment.

15 There is needed a means by which already-running specific instances, particularly of the kind heretofore described, are linked to client applications that reside on an operating server particularly where the object has not registered itself with the operating system. It is important that any such means operate within the bounds of the ideology of maintaining independency between the servers and clients as proscribed by

20 Object Oriented Programming.

## **OBJECTS AND SUMMARY OF THE INVENTION**

It is an object of the present invention to access programming objects.

25 It is another object of the present invention to access specific instances of programming objects.

It is yet another object of the present invention to access unregistered specific instances of programming objects.

It is still another object of the present invention to access already-running programming objects.

It is quite another object of the present invention to access programming objects associated with a PLC.

5 It is further another object of the present invention to access programming objects remotely.

It is indeed another object of the present invention to maintain the object oriented programming methodology of independency of servers and objects.

10 In accordance with the foregoing objectives, the present invention provides means, method, apparatus, system and article of manufacture for accessing specific instances of already-running programming object(s). In one aspect of the present invention, there is provided a manner in which there is accessed a specific instance(s) associated with a PLC coupled to an operating system in the case where the specific instance is not registered with the operating system such that a server is not able to  
15 normally access the specific instances using a registration of the operating system. The invention, upon the determination that the specific instance is not registered, registers the specific instance such that the server is able to randomly access the specific instance.

20 In another aspect of the present invention, objects are accessed via remote access such as the internet, intra-net or other remote access. An additional aspect of the invention accesses objects while maintaining the methodology of programming independency characteristics of object-oriented programming.

These and other objects will be appreciated in light of the following description of the drawings wherein the numerals correspond to like elements.

### **BRIEF DESCRIPTION OF THE DRAWINGS**

Fig. 1 is a flow diagram of the present invention;

Figs. 2A-2G are software listings of the present invention;  
 Fig. 3 illustrates the apparatus of the present invention; and  
 Fig. 4 is a flow diagram of the present invention.

## **DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS**

The present invention relates to a method, system, apparatus and an article of manufacture for accessing computer programs. In a preferred mode, the present invention is practiced in the form of a moniker for locating, activating and (or) initializing program objects particularly in a large system environment and (or) particularly in an environment where the object does not necessarily have a persistent state. It will be appreciated by those skilled in the art that the invention may be practiced in another form. For that matter, this disclosure shall not be construed to limit the present invention to any specific environment (i.e., Microsoft™, Windows NT™, COM environment), but may be applied to any platform, environment or operating system. While the present invention has been described based on the C++ programming language, the disclosure shall not be understood to limit the invention to any particular computer programming language (C++, etc), but may include any language. The present invention may, moreover, be practiced not only as a method or process, i.e., computer program – but as firmware, that is, hardware implementation (Programmable Memory, Integrated Circuit), an article of manufacture (CD or DVD disc, etc.) encoded with software, or a machine or apparatus (processor controlled computer, etc.) as supported by the specific machine implementation set forth in the disclosure.

It is possible, using monikers, to access an already-running (specific instances) of an object. Monikers contain information that allows COM objects to be located, activated and initialized. Actually, monikers are themselves COM objects. From a more technical point of view, a moniker is a composite name for an object that includes a

pointer to the object. Clients invoke a moniker by holding references to it in the form of pointers to the *IMoniker* interface.

The *IMoniker* interface includes a function (*BindToObject*) for binding the moniker to the object to which the pointer of the moniker points. Binding causes an object to be placed in a running state so that the services supplied by the object may be invoked. This interface contains a large number of other methods to cope with different situations, but the key one for this discussion is the method, *BindToObject()* shown here:

```

interface IMoniker : IPersistStream
{
    HRESULT BindToObject([in] IBindCtx *pbc,
                        [in] IMoniker *pmkToLeft,
                        [in] REFIID riidResult,
                        [out] void **ppvResult);
    // other methods removed for clarity

```

The client invokes the *BindToObject()* function of *IMoniker* to execute the binding process. The precise implementation details are - as usual for COM - up to the object that implements *IMoniker*. But in principle, if the object happens to be already running, *BindToObject()* should be able to find it and return an interface pointer on it. If the object is crystallized in a persisted state on a storage medium, it is *BindToObject()* which ought to be able to locate the right server for the class it belongs to, launch this server and ask it to bring it back to life in memory, all without any aid on the client side. The resultant interface pointer of type *riid* (third parameter) is returned in *ppvResult* (fourth parameter), assuming that the process has run smoothly and flawlessly.

Besides the Imoniker interface, the COM specification allows monikers to be defined by a string of text, i.e., a **string moniker**, called the **display name** of the moniker. The display name is made up of two distinct parts separated by the : delimiter. The substring to the left of the : determines the type of moniker, while the substring to the right is highly instance-specific and furnishes a textural version of the state of the object.

However, the display name cannot be translated into a fully functional class instance in a single pass. The display name must first be parsed. This is done by calling the MkParseDisplayName() API (declared in objbase.h) that determines the type of moniker by converting the moniker prefix substring from a ProgId to a CLSID. After that, the IParseDisplayName function is called that first queries the class object of the moniker and, if the response is negative, querying a new instance of the coclass allocated for this purpose. After a valid pointer to IParseDisplayName is obtained, the function passes the entire display string to I parseDisplayName ::ParseDisplayName(), which is supposed to do the hard parsing work and calls the IMoniker function described above that returns an appropriate moniker object.

Monikers save programmers time when coding various types of COM-based functions. The linked document, for example, contains a moniker that identifies its source, so when the user or program activates the linked object to edit it, the moniker is bound. Thus, it becomes possible to load the source into memory without any precise knowledge of where the linked object resides. This is particularly important as it frees clients from being burdened with locating objects.

The ability to access objects easily makes Monikers useful for instantiating classes within the OLE environment (Object Linking and Embedding). Again, OLE allows an object to be embedded into a document, called the container application. An

object can be linked instead of embedded - in which case the container application does not physically hold the object, but provides a pointer to it. In any event, monikers are quite useful for instantiating objects within compound documents.

- 5 Platforms provide built-in moniker types that implement the most recurring binding algorithms. These are summarized briefly in the table below:

<u>Name</u>	<u>Special Requirements</u>	<u>Description</u>
Class Moniker	Windows NT 4.0+	Binds to a class object
File Moniker		Binds to an object persisted on file
Pointer Moniker		Encapsulates a pointer to an active object in a moniker
Composite Moniker		Combination of several monikers
Item Moniker		Sub-object in a composite moniker
Java Moniker	Internet Explorer 4.0+	Exposes the classes exported by the IE4 JVM
URL Moniker	Internet Explorer 3.0+	Encapsulates a URL pointing to a distributed resource
ObjRef Moniker	Win98, Win95 with DCOM, WinNT 4.0 SP4+	Encapsulates a pointer to an object running out-of-process in a moniker
Anti Moniker		Nullifies other monikers in a composite

However, the foregoing monikers require local parameter data to access a specific instance. For example, a specific instance of the class balloon is meaningless without the local parameter that indicates the specific color of this particular balloon. Typically, local parameters are accessed by clients in persistent memory, that is, memory that continues to exist after the program that created it is not available. Problematically, it is not so easy to obtain the persistent data – this is particularly true for large network systems where the persistent data may be located anywhere on the network.

This problem is compounded when it is considered that it is not always the case that objects are responsible for registering themselves on the Running Object Table (ROT). When a client desires to instantiate a specific instance, it fetches the pointer of the persistent data by first looking up the registered object on the ROT. In the case where the running object failed to register itself on the ROT, only that running object has the pointer to the persistent data and the client cannot instantiate that specific instance.

These problems may be better understood by analyzing the following file moniker construct. This file moniker is a moniker that binds to an object persisted on file. More specifically, the file moniker is encapsulated in the API call CoGetObject. Within this function, the file path is identified in a string (c:\MyDirectory\MyFile.ext). If the file is a compound document, that is, a document composed of more than one object, COM may use the API function call GetClassFile to retrieve the **CLSID** (the class ID - a unique number that identifies the type of the object) from the persistent data stored in the file.

Once the class is known, COM checks the ROT to determine if an instance with this persistent state has been registered as running. If the instance is registered, then COM returns the pointer of this object to the client. If the instance is not registered, then

an instance of the object is instantiated and COM queries the object for a particular standard interface (named IPersistFile). If this is successful, the method Load is called with the filename. The object can then load the file and initialize its state. The string that is passed to CoGetObject can have other formats as those skilled in the art will appreciate.

In the case that the object is stored in a container, the OLE fileitem moniker, i.e., c:\MyDirectory\MyFile.ext\item1, is employed. The object named by the file moniker with the string "c:\MyDirectory\MyFile .ext" is a container that can hold some objects, one of which is named "item1". In this case, after the load function is called on the object (or the object has been retrieved from the running object table), the object is queried for the interface IParseDisplayName. If this call is successful then an Item moniker is returned to the COM and composed with the file moniker. The resulting composite moniker, i.e., the combination of the file and the item moniker, is returned to the client and the method BindToObject of the interface Imoniker is called on it.

The bind operation occurs in the reverse order. When the composite moniker is bound, it splits into two parts – the rightmost constituent moniker (the item moniker) and the remaining portion of the composite (in this case the file moniker). BindToObject is then called on the item moniker. The item moniker cannot resolve its name ("item1") without a container. It calls BindToObject on the moniker to the left asking for the interface IOleItemContainer and, if this call is successful, calls the method GetObject (passing its name as the parameter). Since the object named by the file moniker is a container that understands the string "item1", the object is retrieved and returned to the client. A file moniker string can have any number of items allowing the representation of arbitrarily complex hierarchies.

The analysis of the following string will be illuminative of the operation of the afore-described file moniker:

c:\siemens\MyBigMachine.waflvariablesimb0

Identifies item provided by service

Identifies service

File moniker for WinAC active file

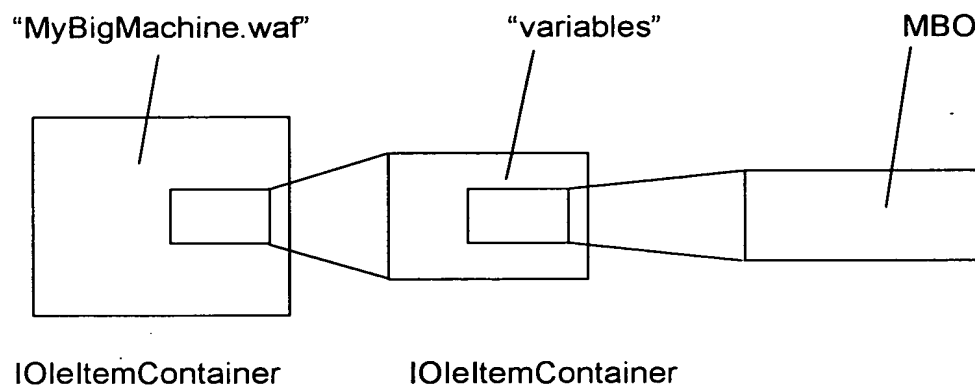
In the example string above, the file moniker uses the string "c:\siemens\MyBigMachine.waf" to locate the server in the ROT. When the object is located, it is queried for the interface IParseDisplayName. A file moniker is created for this object that will be used in the subsequent compose operations.

The ParseDisplayName method of the interface IParseDisplayName is called with the item name "variables" and if the operation is successful an item moniker is created which is then composed with the file moniker created above. The resulting object is then queried for the interface IParseDisplayName and the method ParseDisplayName is called with the item name "MBO". If this is successful, another item moniker is created for the string "MBO" and composed with the composite created in the previous operation. At this time, the string is completely consumed. The resulting composite moniker is returned to the client for binding.

The binding operation begins with the rightmost item moniker. The composite is traversed until reaching the file moniker, where GetObject is called successively with each item name. When the sequence is complete, the object representing MBO is returned to the client. This object can either support a default property containing the

value and/or implement IDataObject. The client can also request only the service "variables" by using the moniker, "c:\siemens\MyBigMachine.waflvariables".

The Data.Ocx (a component of the SIEMENS product WinAC) would use this moniker to retrieve an IVar session object.



This naming convention can support any level of granularity. It would be possible, for example, to name elements in data blocks c:\siemens\MyBigMachine.waflvariables (i.e. idb51elt4).

Since the foregoing naming convention will support an arbitrarily complex hierarchy, it could be used to name devices and input/output (I/O) points in the field. As new services are added to the various WinACTM components, no change would have to be made to existing clients to use them. All of the servers will support the same basic syntax for the moniker strings. However, since the servers themselves are moniker providers, each server can extend the syntax as needed.

A client may want to connect to a specific instance of an object that has no persistent state and therefore cannot register a File moniker. The moniker of the present invention resolves this problem. In summary, this moniker is a software

component that allows a server to register an instance of an object in the ROT with a user-defined name. The user-defined name can be thought of as an item moniker. After an instance of an object is registered in the ROT, that instance is then accessible from any client. If a client wants to access the object, it uses the Running moniker to search the ROT for the instance and bind to it. A server, thus, can register any number of object instances, each with a different name. When a server's execution is terminated, it should unregister any object instance that it registered.

The steps of the present invention will now be described with reference to Fig. 1. The first step (S100) is to register the instance. This accomplished by calling the following methods provided in the IRunning interface:

```
HRESULT RegisterInstanceName (BSTR bstrItemName, IUnknown* pUnk, long*
ICookie);
HRESULT UnregisterInstanceName (long ICookie).
```

As a result of the function call, the server passes both a pointer to the instance and the desired name to the RegisterInstanceName method. If this call is successful, then a cookie is returned. The server caches the cookie, which is used when the object is destroyed as the parameter to the method UnregisterInstanceName. If a server fails (i.e., it crashes) to unregister the object upon its destruction, the ROT purges the object's moniker on the first attempt to access it.

The moniker of the present invention implements the standard interface IParseDisplayName (S102). A client can locate a named instance of an object by calling either GoGetObject or MkParseDisplayName with the following string "@Running:ObjectName". At this point, COM converts the ProgID "Running" to the CLSID using the API call CLSIDFromProgID. Next, the API call GoGetClassObject is

used to instantiate the parser. The object is then queried for the IParseDisplayName interface. The method ParseDisplayName is called on that interface to parse the string (S103). If the requested object cannot be located in the ROT, an error is returned to the client (S104). At this time, a pointer moniker is created (S105) and returned to the client (S107). The client then calls BindToObject on the returned moniker. When a client is finished using the object it releases it.

If the object registered in the ROT supports the interfaces IParseDisplayName and IOleItemContainer, more complex operations are applied (S106). The moniker of the present invention is capable of querying the object for IParseDisplayName and calling the method ParseDisplayName with the name of an item (S108). If this call is successful (S109), the returned object can also be queried for IParseDisplayName. If not successful, a failure code is returned (S110). If the object supports this interface, the name of the object is used to create an item moniker which is composed (S111) with the moniker to the right.

The above steps can be continued recursively for an arbitrary complex hierarchy. Once all of the elements of the input string have been consumed, the resulting composite moniker is returned to the client for binding (S112). The bind process BindToObject is called with the interface IOleItemContainer, recursively calling the method GetObject until the composite moniker is consumed and the object is returned to the client.

The standard moniker process uses the Microsoft defined interfaces, IOleItemContainer and IParseDisplayName, and creates only instances of Item and Pointer monikers for use during the binding process. The moniker of the present invention is not WinAC specific and has no dependencies on any product. It can be used for any object that wants to be registered in the ROT. The ability to use the

IParseDisplayName and IOleItemContainer interfaces is modeled after the File moniker and will work with any object that supports these interfaces. Their use, however, is optional and the moniker of the present invention can be used only to register and retrieve objects from the ROT.

5

Thus, the present invention provides a way to name objects that would otherwise stay unnamed. After that, the object can be called (i.e., located) by name. Figs. 2A-2G set forth the C++ software implementation of the present invention, wherein:

10 Fig. 2A: Running.idl

Contains the definition of the COM interfaces provided by the Running moniker.

The interface defined by the Running moniker is:

IRunning: An interface derived from the standard interface I dispatch

15 The object "Running Moniker" provides the following interfaces:

IRunning the interface defined within the file

I ParseDisplayName a standard interface defined by Microsoft

IUnknown a standard interface defined by Microsoft

20 Fig. 2B: Running.rgs

This file contains the necessary registry entries for the Running Moniker.

25 Fig. 2C: Crunning.h

This file contains the declaration of the C++ class, which implements the functionality of the Running Moniker.

Figs. 2D-2G: Crunning.

This file contains the actual implementation of the Running Moniker.

The methods Register instanceName and Unregister instanceName are used to put in or remove the name of a running object into or from the table of running objects.

The method ParseDisplayName searches the name of a registered object within this table and returns an object, which points to this object.

It will be appreciated that the present invention may be practiced in other programming languages, or as an apparatus or article of manufacture as shown in Fig.

3. In more detail, Fig.3 shows the system 300 of the present invention wherein a computer 302 (such as a personal computer (PC) ) includes a processor 304, display 306 (and display connection 308), non-volatile memory 310A, B, remote connection(s) 312, ISA/MPI card(s) 314 and PLC(s) 316 (and connections 318). The moniker of the present invention may reside, as an article of manufacture, on the non-volatile memory 310A, B such as the disc(s) 310A (floppy, CD, DVD, etc.) or resident memory (hard disc, cache memory) 310B. The PC 301 is connected to PLC(s) 316, via a remote connection 312 (USB, COM, Internet, Intranet, Network, RS-232, etc.). The PLCs 316 may be daisy-chained together by a bus 318 in a master/slave relationship, for example. In another aspect, the PLC(s) may be interfaced with firmware 314 (ISA/MPI cards, etc.) that are provided by, for example, WinAC or Simatec whose function is to interface and communicate with the remote PLC(s) 316. The PLC(s) may also be provided in the form of firmware 314 installed in the slots.

It is possible that the moniker of the present invention be practiced as a **dynamic link library** (d11) that communicates through the communication ports (particularly, the USB port). In that case, the present invention may incorporate a well-known device driver (which one skilled in the art will appreciate how to implement) to achieve a high degree of communication between the processor 304 and the PLC(s) 316. For

example, the connection may be the USB port and the driver may be an ActiveX™ device driver that drives the processor. One of the advantages of this arrangement is that the processor is driven directly, i.e., without the need for firmware.

5 In operation, the system in one aspect of Fig. 3 loads the moniker of the present invention from the non-volatile memory 310A,B. The processor 304, in order to access a PLC object, invokes the moniker of the present invention. The moniker retrieves the name of the desired PLC, either directly through the remote connection 312 or as provided by the interface of the firmware 314 that communicates with the PLC(s) 316.

10 For example, an Excel™ client application instantiates the specific instance of the master PLC 316 using the moniker of the present invention. At this time, the Excel™ client has the pointer to the master PLC and may pass parameters therefrom/thereto. For example, the Excel™ client may retrieve operating data from the PLC(s) 316 and display the same in Excel™ format on the display 306. This last example is particularly

15 useful where the PLC(s) do not otherwise have a convenient means of displaying data. It will be appreciated that, since the moniker of this present invention activates already-running objects, there is a security measure that an otherwise dormant PLC will not be erroneously activated which could disastrously effect connected machinery (i.e., motors), not shown.

20 The present invention is useful for instantiating specific instances of a soft PLC application such as provided by Simatic or WinAC (Siemens proprietary hardware/software). In such applications, the PLC may be unable to register itself in the operating system. The PLC(s) may not be able to register itself because it is remotely

25 connected, for example, through a remote connection such as MPI, Universal Serial Bus, COM port, serial port (RS-232) or the like. The PLC may also be installed as firmware on a card such as MPI or ISA which has no traditional means to register objects. This is significant in a system connected to a plurality of PLCs, because it is

necessary for the server to determine the specific PLC to access. Accessing the wrong PLC in a real-world environment could be disastrous

In one particular soft PLC environment, the WinAC environment (proprietary software/hardware provided by Siemens), the moniker of the present invention is utilized to access specific PLC objects. In particular, it is a problem in WinAC to name service providers. At the core of the problem, WinAC uses the moniker "@WinAC". Problematically, the WinAC moniker names an implementation of the interface IVar rather than an instance. The moniker "@WinAC:default", while a more generic approach, assumes that only one IVar provider is located on a machine. While at the present time only one IVar provider can run on a machine (either WinAC or the SlotPLC), it is undesirable to limit the system to only one provider.

In addition, the names of the implementations are part of the moniker. With the previous strategy, there is no way for the user to apply meaningful names (i.e. MyBigMachine) to objects in the system. If the user decides that a different implementation is required (for example, switch between WinAC and SlotPLC), the user must change the name of the server in the tagfile. This means that the user has to have the STEP7 (proprietary Siemens software/hardware) projects and be able to recreate the tagfile. The tagfile is a database within WinAC which stores the relationship between symbolic names (meaningful to the user) and absolute addresses within the process.

There is also a problem with object identity. When a client asks for "@WinAC", an IVar "session" object is returned. Problematically, each client has its own instance of IVar session object. This causes confusion because the use of the moniker implies that a client is connecting to a specific instance. The "session" objects are required to maintain state information such as server handles. Since the "session" objects maintain

a state, they are not interfaces and some other means (namely, the present invention) should be provided to create them.

The moniker of the present invention resolves the foregoing problem of naming WinAC service providers by establishing a means by which the user names instances of objects. Using the named instance method disclosed herein, it would be possible to use the tagfile entry "MyBigMachine, MBO" to refer to MBO of whatever IVar server is named "MyBigMachine". This allows the user to change implementations without needing to change the tagfile. Indeed, any number of uniquely named instances can run simultaneously on one machine. Thus, clients can use the moniker of the present invention to connect to the correct instance.

The following example of the string "Running:MyBigMachineIvariablesImb0", as described with reference to Fig. 4, illustrates the manner in which the moniker of the present invention can be utilized in the WinAC situation.

Running:MyBigMachineIvariablesImb0

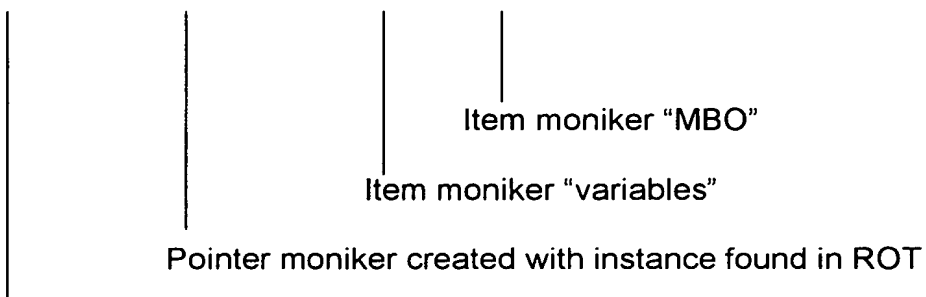
The diagram shows the string "Running:MyBigMachineIvariablesImb0" with vertical lines indicating its structure. From left to right, the components are: "Running:", "MyBigMachine", "Ivariables", and "Imb0". Labels with arrows point to these components: "Loads running moniker" points to "Running:", "Identifies named instance" points to "MyBigMachine", "Identifies service" points to "Ivariables", and "Identifies item provided by service" points to "Imb0".

The moniker of the present invention uses the string "MyBigMachine" to locate the server in the ROT (Step S400). When the object is located (Step S401), it is queried for the interface IParseDisplayName (Step S402). A pointer moniker is then

created for this object that will be used in the subsequent compose operations (Step S404). The ParseDisplayName method of IOItemContainer is called with the item name "variables" (Step S406). If the operation is successful (Step S407), an item moniker is created which is then composed with the pointer moniker created above (Step S408). The resulting object is queried for IParseDisplayName and the method ParseDisplayName is called with the string "ImbO" (Step S410). With the exception of using a pointer moniker for the leftmost moniker in the bind operation (Step S412), the sequence of operations of the File moniker already described may be implemented hereafter.

The resulting moniker string definition of the WinAC implementation is shown below.

"Running:MyBigMachine\variables\ImbO"



Running moniker searches ROT for "MyBigMachine"

As will be understood from the foregoing string, the moniker of the present invention provides a pointer moniker for each partition of the string. In the situation where the object is a soft PLC object, the name of the object comes from an address specially allocated by the soft PLC. In WinAC, for example, the name comes from the MPI address which is allocated by the processor. The name may also come from the DMA channel, memory address, IRQ, or COM port.

It will be appreciated that the moniker in one aspect of the present invention connects to already running objects. Thus, a security measure, this ensures that an otherwise dormant PLC is not activated erroneously. The present invention provides a flexible and robust method for naming objects with monikers. The correct usage of these objects provided by the present invention allows for an extensible solution to activation problems. If a standard implementation is applied across all WinAC components, it will be easy to put the information needed to connect to a server in a database such as the tagfile. By making the servers into moniker providers, the problem of coordinating updates for parsers as components are changed (created by making the moniker parsers intelligent) will be avoided. In the model described, the moniker of the present invention does not require any knowledge of the various WinAC components. In any case, by having a consistent approach and eliminating special cases, problems are avoided in any environment.